12/20/24

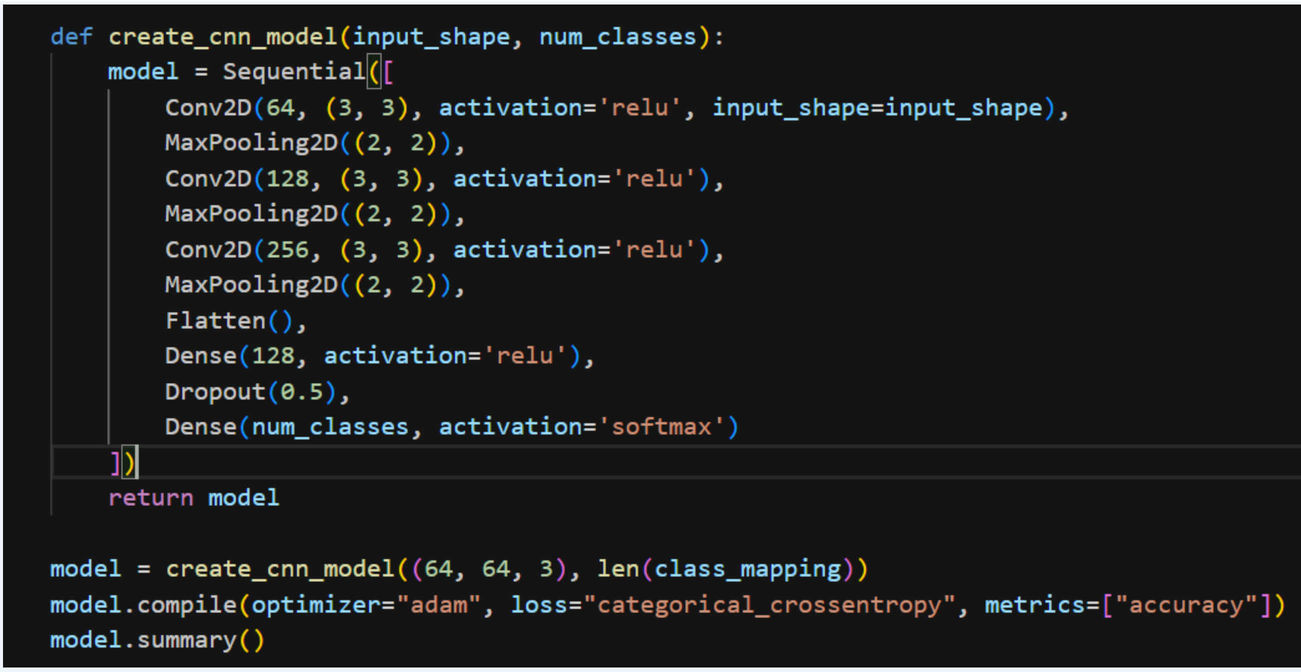
Kaggle Animals Classification Report
IronHack Miami
AI Engineering

Alex Castillo
Victor Rivera
Gerardo Gonzalez

- **Description of the chosen CNN architecture.**

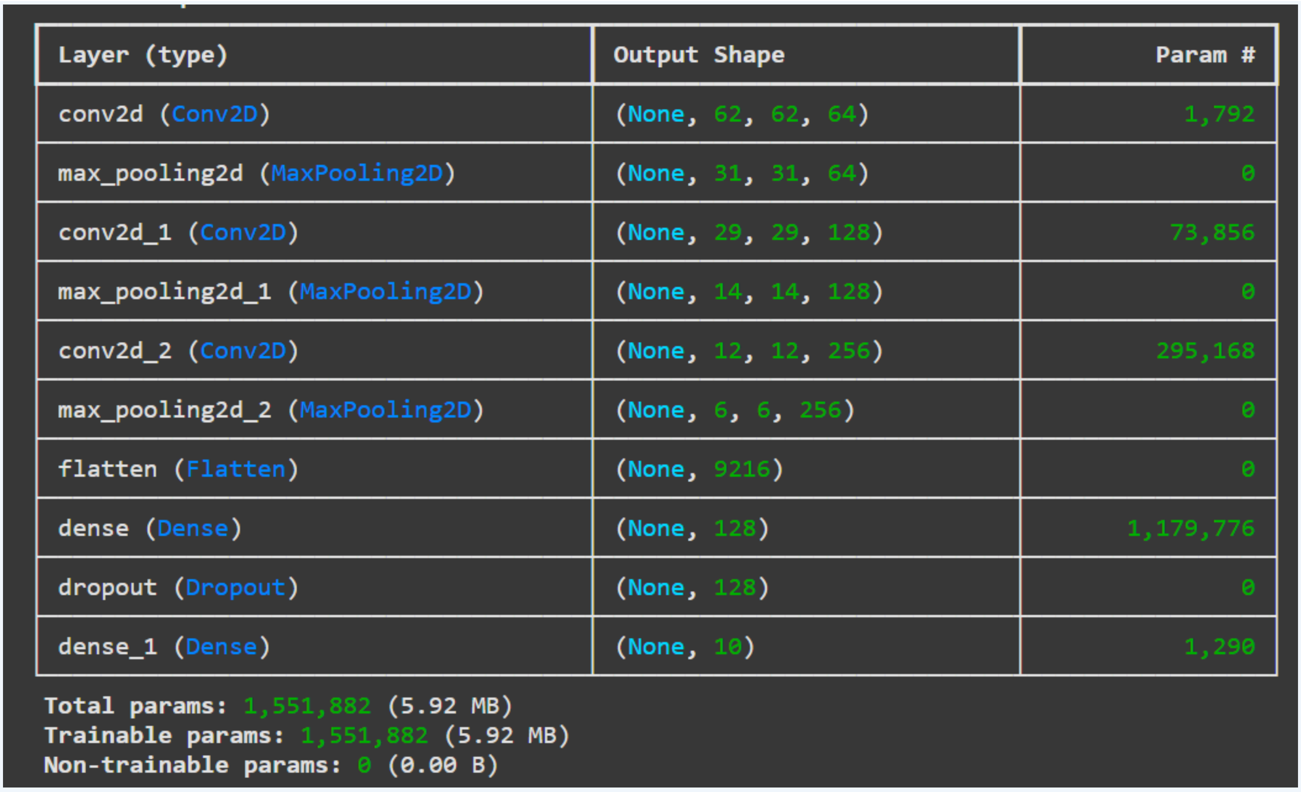
We tested **2 architectures** :

1. A **Convolutional Neural Network (CNN) Sequential** model



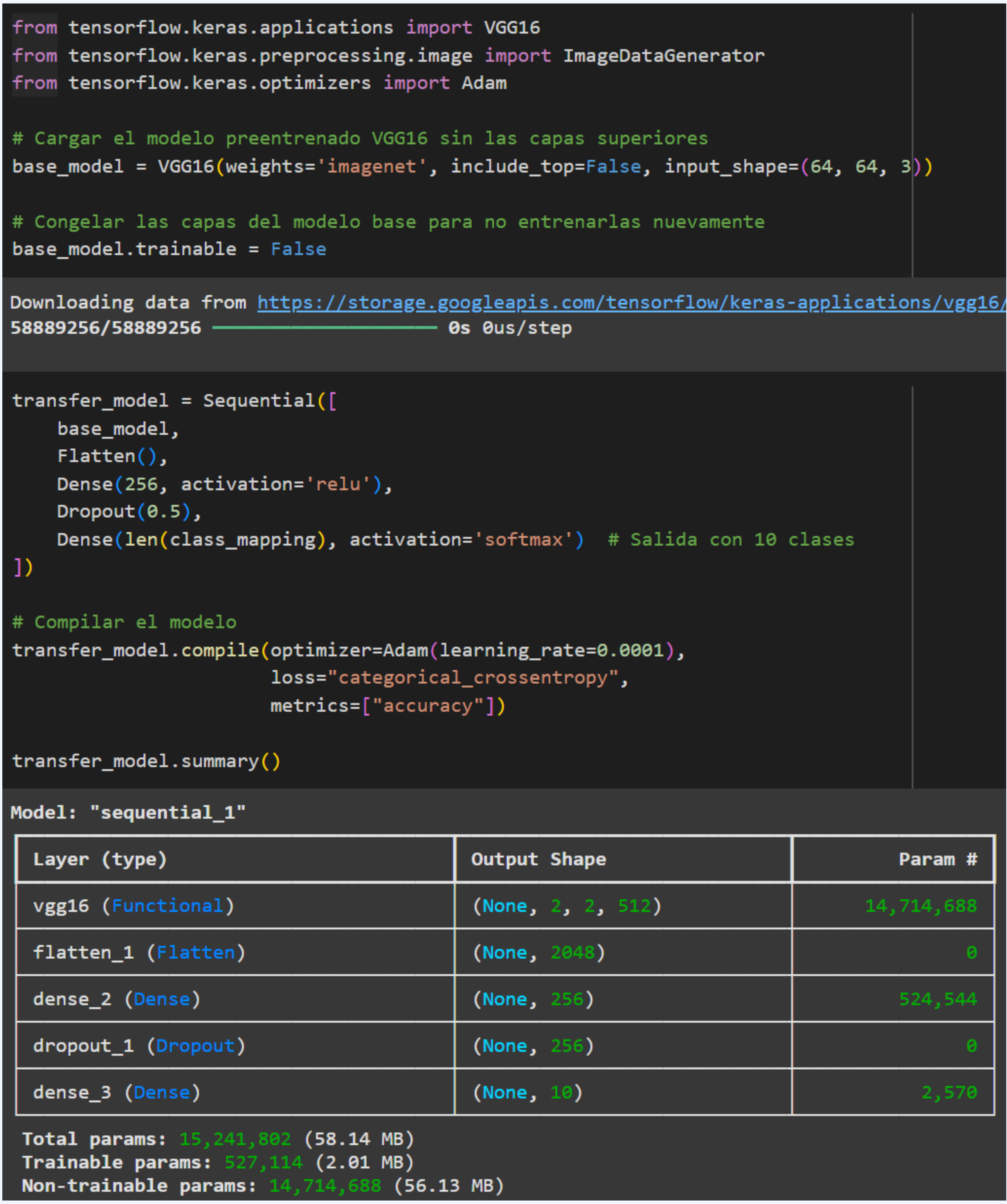
```python
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    return model

model = create_cnn_model((64, 64, 3), len(class_mapping))
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
```



```
┌──────────────────────────────────────┬─────────────────────────────┬──────────────────┐
│ Layer (type)                         │ Output Shape                │          Param # │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d (Conv2D)                      │ (None, 62, 62, 64)          │            1,792 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 31, 31, 64)          │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 29, 29, 128)         │           73,856 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 14, 14, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 12, 12, 256)         │          295,168 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 6, 6, 256)           │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ flatten (Flatten)                    │ (None, 9216)                │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense (Dense)                        │ (None, 128)                 │        1,179,776 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout (Dropout)                    │ (None, 128)                 │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense_1 (Dense)                      │ (None, 10)                  │            1,290 │
└──────────────────────────────────────┴─────────────────────────────┴──────────────────┘
 Total params: 1,551,882 (5.92 MB)
 Trainable params: 1,551,882 (5.92 MB)
 Non-trainable params: 0 (0.00 B)
```

2. A **Transfer Sequential model from VGG16 base model**:



```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam

# Cargar el modelo preentrenado VGG16 sin las capas superiores
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64, 3))

# Congelar las capas del modelo base para no entrenarlas nuevamente
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
transfer_model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(len(class_mapping), activation='softmax')  # Salida con 10 clases
])

# Compilar el modelo
transfer_model.compile(optimizer=Adam(learning_rate=0.0001),
                       loss="categorical_crossentropy",
                       metrics=["accuracy"])

transfer_model.summary()
```

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ vgg16 (Functional)                   │ (None, 2, 2, 512)           │      14,714,688 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_1 (Flatten)                  │ (None, 2048)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ (None, 256)                 │         524,544 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 256)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_3 (Dense)                      │ (None, 10)                  │           2,570 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 15,241,802 (58.14 MB)
 Trainable params: 527,114 (2.01 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
```

- **Explanation of preprocessing steps**.
  Since there were no problematic items on the dataset, the only preprocessing we did was turn the scientific names of the folders to English common language.

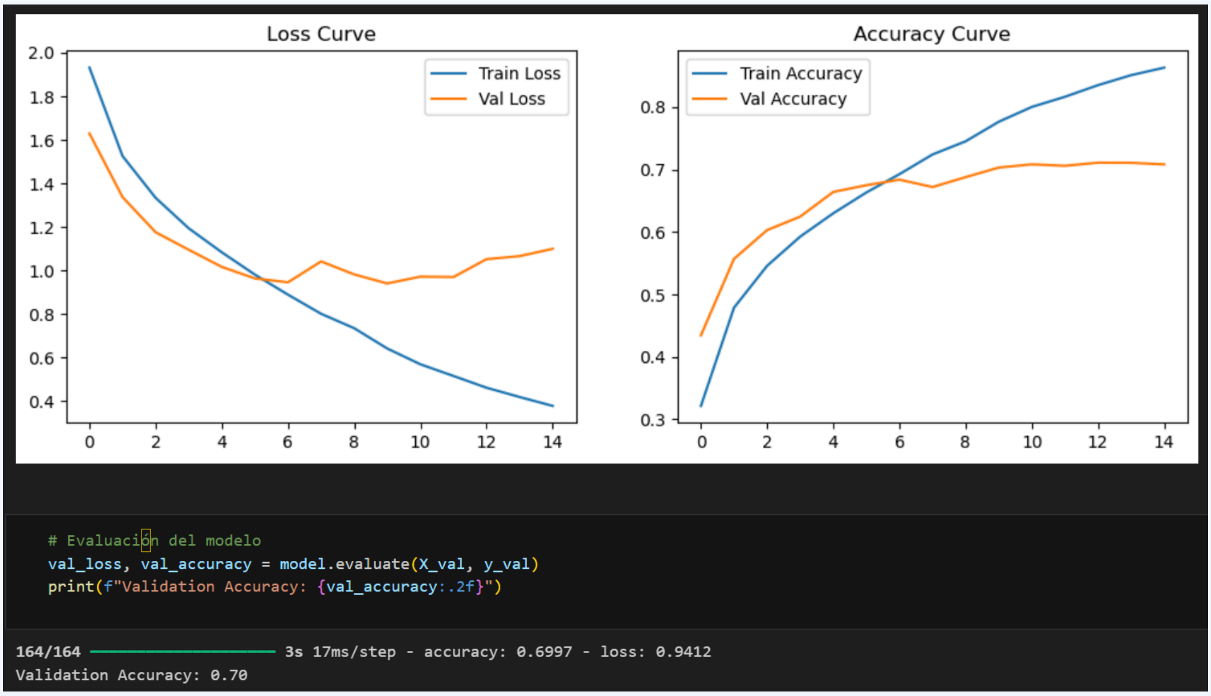
  From there we built the dataset by obtaining the **images** from their path and the **class names** and separating them in **X** and **y** variables. The class names **indexes** were used as the **target**. Then the **X** and **y** were fitted into the models

- **Details of the training process (e.g., learning rate, batch size, number of epochs).**
  1. For the CNN model we use 64 inputs and 3 convolutional layers of 64, 128 and 256 filters with max pooling layers in between them. It follows with a dense layer of 128 nodes with an activation function of softmax. We compiled them with Adam optimizer, categorical_crossentropy for loss and accuracy metrics. We trained the model using a batch of 64 and 25 epochs.

  2. For the Transfer model we use a base model of 64 inputs. We transfer it to a Sequential model of a dense Layer of 256 filters with activation function ReLu. We dropout 0.5 and then a Dense layer output of the amount of classes activating with softmax. We compiled them with Adam optimizer with a learning rate of 0.0001 , categorical_crossentropy for loss and accuracy metrics. We trained the model using a batch of 64 and 25 epochs.

- **Results and analysis of models performance.**

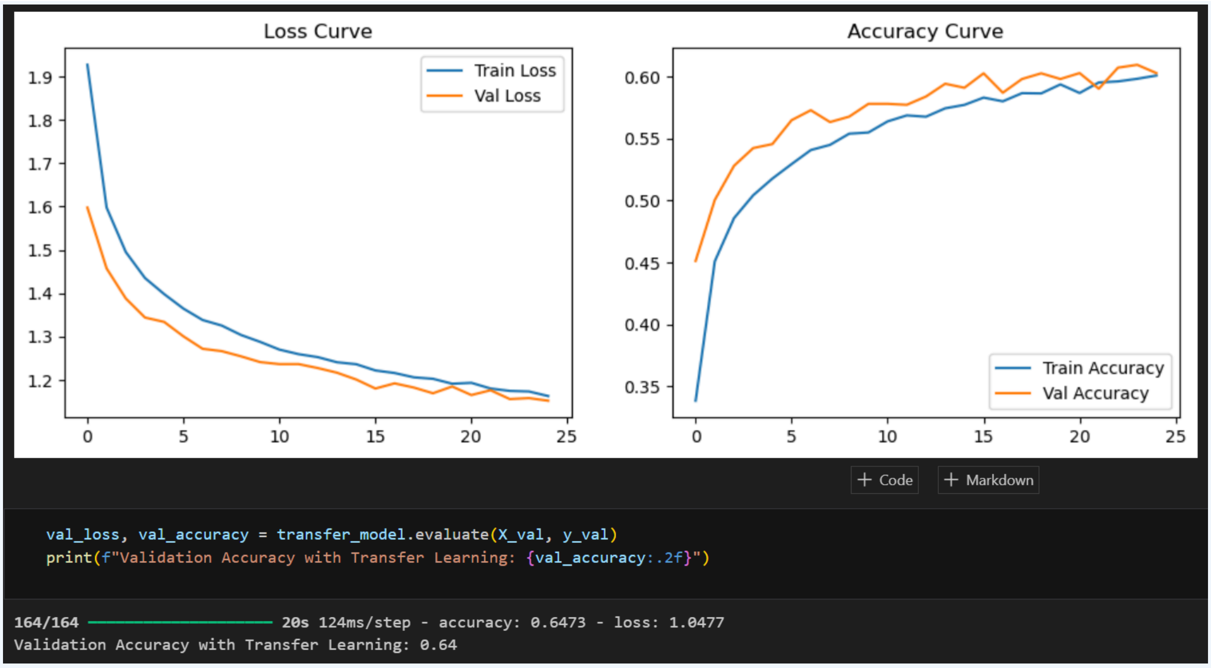
1.



```
# Evaluación del modelo
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy:.2f}")
```

```
164/164 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.6997 - loss: 0.9412
Validation Accuracy: 0.70
```

2.



```
val_loss, val_accuracy = transfer_model.evaluate(X_val, y_val)
print(f"Validation Accuracy with Transfer Learning: {val_accuracy:.2f}")
```

```
164/164 ━━━━━━━━━━━━━━━━━━━━ 20s 124ms/step - accuracy: 0.6473 - loss: 1.0477
Validation Accuracy with Transfer Learning: 0.64
```

- **What is your best model? Why?**

  The best model was the **Convolutional Neural Network (CNN) Sequential Model** because it had a higher precision in accuracy and less Total Loss.